TP 7 – ALGORITHME DE TRIS D'UNE LISTE

Préambule :

1. Ouvrir une session sur l'ordinateur en entrant Identifiant / Mot de passe ;
2. Lancer Spyder en cherchant Spyder dans la barre de recherche ;
3. Effacer le texte dans la fenêtre de gauche et écrire :

```
#NOM
#PRENOM
#TP7
```

Objectifs de ce TP :

- *Tri par sélection ;*
- *Tri par insertion ;*
- *Tri à bulles ;*
- *Tri rapide.*

Toutes les réponses aux questions doivent écrites dans le programme qui sera à rendre dans le dossier Restitution de devoirs de votre classe avant 18h.

Rappels de cours sur l'utilisation des listes :

- Pour construire une liste, on énumère les éléments de la liste entre crochets, séparés par des virgules :

- Par exemple : L=[3,7,42,1,4,8,12]

- Il est également possible de créer une liste par concaténation de 2 listes :

```
c=[0,1]+[2,4]      #résultat : c=[0,1,2,4]
d=3*[0,1]          #résultat : d=[0,1,0,1,0,1]
```

- Les opérations « + » et « * » ne réalisent donc pas l'addition ou la multiplication élément par élément.

- Les éléments d'une liste sont numérotés ou indicés à partir de 0 : « L[2] » correspond donc au 3e élément de la liste L.

- Pour modifier la deuxième case de a et lui donner la valeur « 0 », il suffit d'écrire : « L[1] = 0 ».

- La fonction « len(L) » permet de connaître le nombre d'éléments de L.

- On peut aussi facilement accéder aux derniers éléments d'une liste : « L[len(L)] » ou « L[-1] » correspond au dernier élément de L par exemple.

- On peut également extraire tous les éléments situés entre les indices i (inclus) et j (exclus) d'une liste a avec la notation « a[i:j] ». Cette notation crée donc une liste de longueur j-i.

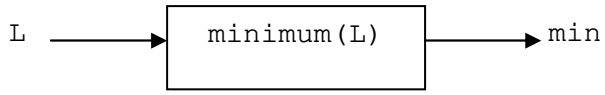
```
a=[3,7,42,1,4,8,12]
b=a[2:5]              #résultat : b = [42,1,4]
```

Dans tous le TP, nous utiliserons des listes aléatoires d'entiers entre 0 et 20 en tapant les lignes de code suivantes :

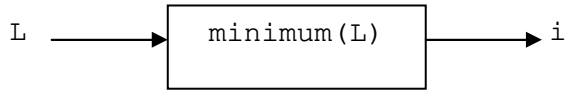
```
from random import *
T=[i for i in range (20)]
print(T)
shuffle(T)
print(T)
```

Exercice 1 : recherche du minimum d'une liste

1. Ecrire à la suite du code le commentaire `#Exercice 1`.
2. Ecrire une fonction `minimum(L)` qui renvoie la valeur du minimum de la liste :



3. Tester votre fonction sur la liste `T`.
4. Modifier la fonction `minimum(L)` pour renvoyer le rang `i` du minimum dans la liste `L`:



5. Tester votre fonction sur la liste `T`.

Exercice 2 : tri par sélection

Le principe du tri par sélection est expliqué sur la vidéo suivante à regarder et re-regarder :

https://www.youtube.com/watch?v=8u3Yq-5DTN8

L'algorithme de tri par sélection va donc utiliser la fonction minimum créée à l'exercice 1 ainsi que la fonction `echange(L,i,j)` décrite ci-dessous qui permet d'échanger 2 valeurs dans une liste :

```
# echange 2 valeurs au rang i et j d'une liste L
def echange(L,i,j):
    L[i],L[j] = L[j], L[i]
```

1. Ecrire à la suite du code le commentaire `#Exercice 2.`
2. Ecrire et tester la fonction `echange(L,i,j)` sur la liste T.
3. Compléter le pseudo-code ci-dessous de la fonction `triselection(L)` qui doit trier une liste L :

```
Définir triselection(liste) :
    Pour k variant de 0 à la longueur(liste) :

        _____________________________

        _____________________________

        Afficher(L) // pas obligatoire : pour voir l'algorithme tourner

    Renvoyer(liste)
```

4. Coder le pseudo-code.
5. Tester votre fonction sur la liste T.

Exercice 3 : tri par insertion

Le principe du tri par insertion est présenté sur la vidéo suivante à regarder et re-regarder :

https://www.youtube.com/watch?v=bRPHvWgc6YM

1. Ecrire à la suite du code le commentaire `#Exercice 3.`

2. Compléter le pseudo-code ci-dessous de la fonction `triselection(L)` qui doit trier une liste L :

```
Définir triinsertion(liste) :
    Pour k variant de 1 à la longueur(liste) :

        Stocker l'élément k de la liste dans la variable cle

        _____________________________

        Tant que (j >0 et _____________________________ > cle) :

            ____________________________________________________

            Enlever 1 à j

        Stocker l'élément j dans la variable cle

        Afficher(L) // pas obligatoire : pour voir l'algorithme tourner

    Renvoyer(liste)
```

3. Coder le pseudo-code.
4. Tester votre fonction sur la liste T.

Exercice 4 : le tri à bulles

Une illustration du tri à bulles est disponible sur la vidéo suivante à bien regarder pour comprendre le principe :
https://www.youtube.com/watch?v=h00xRb-L8Zs&t=2s

Le tri à bulles est un algorithme simple à comprendre :

- on parcourt toute la liste en permutant 2 à 2 les éléments s'ils ne sont pas dans le bon ordre : **l'élément le plus grand « remonte » ainsi en dernière position** ;
- on reproduit l'opération, en se limitant aux $n-1$ premiers éléments, **s'ils sont encore en désordre**.

1. Ecrire à la suite du code le commentaire `#Exercice 4.`
2. Compléter le pseudo-code ci-dessous de la fonction `triabulles(L)` qui doit trier une liste L :

```
Définir triabulles(liste) :
    Pour k variant de 1 à la longueur(liste) :

        ____________________________________________________

            Si l'élément j est > à l'élément j+1 :

                ____________________________________

                Afficher(L) // pas obligatoire

    Renvoyer(liste)
```

1. Coder le pseudo-code.
2. Tester votre fonction sur la liste T.

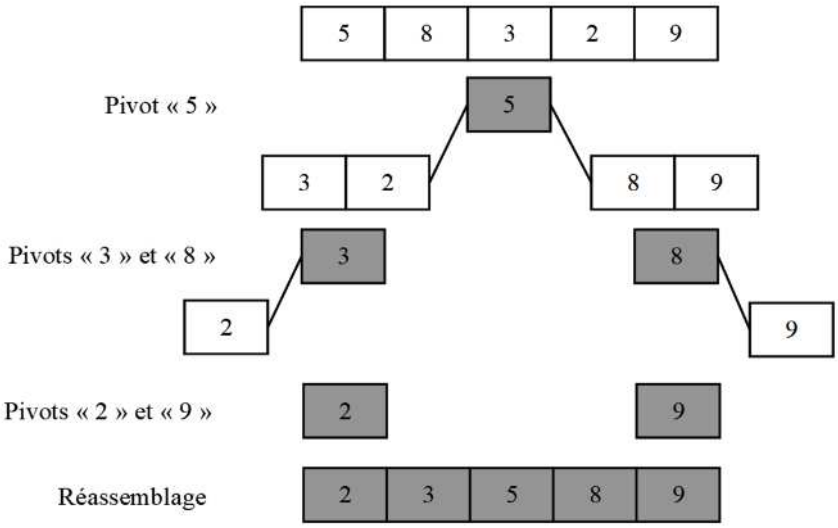
Exercice 5 : le tri rapide : diviser pour régner

L'algorithme de tri rapide fait partie de la catégorie des algorithmes diviser pour régner. A chaque appel de la fonction de tri, le nombre de données à traiter est diminué de un. C'est-à-dire que l'on ne traite plus l'élément appelé pivot dans les appels de fonction ultérieurs, il est placé à sa place définitive dans le tableau.
Le tableau de valeurs est ensuite segmenté en deux parties :

- dans un premier tableau, toutes les valeurs numériques sont inférieures au pivot,
- dans un second tableau, toutes valeurs numériques sont supérieures au pivot.

L'appel de la fonction de tri est récursif sur les tableaux segmentés. C'est-à-dire qu'il fait appel à lui-même.

Un exemple est donné ci-dessous sur une petite liste :



Un exemple plus complet est donné sur la liste suivante :
[4, 11, 15, 13, 14, 9, 6, 0, 12, 17, 18, 3, 1, 10, 7, 16, 2, 19, 8, 5]}
#liste des petits liste des grands
[0, 3, 1, 2] [11, 15, 13, 14, 9, 6, 12, 17, 18, 10, 7, 16, 19, 8, 5]
[] [3, 1, 2]
[1, 2] []
[] [2]
[] []
[9, 6, 10, 7, 8, 5] [15, 13, 14, 12, 17, 18, 16, 19]
[6, 7, 8, 5] [10]
[5] [7, 8]
[] []
[] [8]
[] []
[] []
[13, 14, 12] [17, 18, 16, 19]
[12] [14]
[] []
[] []
[16] [18, 19]
[] []
[] [19]
[] []

#liste finale
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

1. Ecrire à la suite du code le commentaire `#Exercice 5.`
2. Compléter le pseudo-code ci-dessous de la fonction `trirapide(L)` qui doit trier une liste L :

```
Définir trirapide(liste) :
      Stocker la première valeur de la liste dans la variable pivot

      Créer             vide

      Créer             vide

      Pour k variant de 1 à la longueur de liste :

            Si l'élément k de la liste est < pivot :

                  _____________________________________________

            Sinon :

                  _____________________________________________

      Afficher(listebasse,'   ',liste haute) // pas obligatoire

      Renvoyer(trirapide(listebasse)+pivot+trirapide(listehaute))
```

3. Coder le pseudo-code.
4. Tester votre fonction sur la liste T.